AnaGate Programmer's Manual

Analytica GmbH

A. Schmidt, Analytica GmbH
S. Welisch, Analytica GmbH

AnaGate Programmer's Manual

Analytica GmbH
by A. Schmidt and S. Welisch

This document was generated with DocBook at 2010-09-09 17:22:53.

Hilfe-Datei (dtsch.): *AnaGate-API.chm*

Hilfe-Datei (engl.): *AnaGate-API-EN.chm*

PDF-Datei (dtsch.): *AnaGate-API-1.10.pdf*

PDF-Datei (engl.): *AnaGate-API-1.10-EN.pdf*

Publication date 09. September 2010
Copyright © 2007-2010 Analytica GmbH

Abstract

The AnaGate Programmer's Manual includes the exact description of the programming interfaces to all models of AnaGate hardware series.

All rights reserved. All the information in this manual was compiled with the greatest of care. However, no warranty can be given for it.

No parts of this manual or the program are to be reproduced in any way (printing, photocopying, microfilm or any other process) without written authorisation. Any processing, duplication or distribution by means of any electronic system is also strictly prohibited.

You are also advised that all the names and brand names of the respective companies mentioned in this documentation are generally protected by brand, trademark or patent laws.

Analytica GmbH
Vorholzstraße 36
76137 Karlsruhe
Germany
Fon +49 (0) 721-43035-0
Fax +49 (0) 721-43035-20

Email <support@analytica-gmbh.de>
www.analytica-gmbh.de [http://www.analytica-gmbh.de]
www.anagate.de [http://www.anagate.de]

Table of Contents

Introduction vii
I. AnaGate API 1
1. The Programming interface of AnaGate devices 3
2. Notes concerning the communication protocol TCP 5
2.1. Important properties of the network protocol 5
3. Common function reference 7
DLLInfo 8
4. CAN API reference 9
CANOpenDevice, CANOpenDeviceEx 10
CANCloseDevice 12
CANSetGlobals 13
CANGetGlobals 15
CANSetFilter 17
CANGetFilter 18
CANSetTime 19
CANWrite, CANWriteEx 20
CANSetCallback, CANSetCallbackEx 22
CANReadDigital 24
CANWriteDigital 26
CANRestart 27
CANDeviceConnectState 28
CANStartAlive 29
CANErrorMessage 30
5. SPI API reference 31
SPIOpenDevice 32
SPICloseDevice 34
SPISetGlobals 35
SPIGetGlobals 37
SPIDataReq 39
SPIReadDigital 41
SPIWriteDigital 43
SPIErrorMessage 44
6. I2C API reference 45
I2COpenDevice 46
I2CCloseDevice 48
I2CReset 49
I2CRead 50
I2CWrite 51
I2CSequence 52
I2CReadDigital 54
I2CWriteDigital 56
I2CErrorMessage 57
6.1. Programming I2C EEPROM 58
7. Programming examples 63
7.1. Programming language C/C++ 63
7.2. Programming language Visual Basic 6 64
7.3. Programming language VB.NET 67
A. Return Codes 69
B. I2C slave address formats 71
C. FAQ - Frequent asked questions 73
D. Technical support 75

© 2007-2010 Analytica GmbH

Bibliography 76

© 2007-2010 Analytica GmbH

List of Figures

7.1. Input form of SPI example (VB6) 64
B.1. Definition of a I2C slave address in 7-bit format 71
B.2. Definition of a I2C slave address in 10-bit format 71

© 2007-2010 Analytica GmbH

List of Tables

1.1. Library files for Windows 3
1.2. Library files for Linux 3
2.1. AnaGate devices and related port numbers 5
6.1. Usage of the CHIP-Enable Bits of I2C EEPROMs 58
A.1. Common return values for all devices of AnaGate series 69
A.2. Return values for AnaGate I2C 69
A.3. Return values for AnaGate CAN 69
A.4. Return values for AnaGate Renesas 70
B.1. I2C EEPROM addressing examples 71

© 2007-2010 Analytica GmbH

Introduction

The AnaGate Programmer's Manual includes the exact description of the programming interfaces to all models of AnaGate hardware series.

The existing interfaces will be decribed below:

- Application Programming Interface (Part I, “AnaGate API”)

© 2007-2010 Analytica GmbH

Part I. AnaGate API

Table of Contents

1. The Programming interface of AnaGate devices 3
2. Notes concerning the communication protocol TCP 5
    2.1. Important properties of the network protocol 5
3. Common function reference 7
    DLLInfo 8
4. CAN API reference 9
    CANOpenDevice, CANOpenDeviceEx 10
    CANCloseDevice 12
    CANSetGlobals 13
    CANGetGlobals 15
    CANSetFilter 17
    CANGetFilter 18
    CANSetTime 19
    CANWrite, CANWriteEx 20
    CANSetCallback, CANSetCallbackEx 22
    CANReadDigital 24
    CANWriteDigital 26
    CANRestart 27
    CANDeviceConnectState 28
    CANStartAlive 29
    CANErrorMessage 30
5. SPI API reference 31
    SPIOpenDevice 32
    SPICloseDevice 34
    SPISetGlobals 35
    SPIGetGlobals 37
    SPIDataReq 39
    SPIReadDigital 41
    SPIWriteDigital 43
    SPIErrorMessage 44
6. I2C API reference 45
    I2COpenDevice 46
    I2CCloseDevice 48
    I2CReset 49
    I2CRead 50
    I2CWrite 51
    I2CSequence 52
    I2CReadDigital 54
    I2CWriteDigital 56
    I2CErrorMessage 57
    6.1. Programming I2C EEPROM 58
7. Programming examples 63
    7.1. Programming language C/C++ 63
    7.2. Programming language Visual Basic 6 64
    7.3. Programming language VB.NET 67
A. Return Codes 69
B. I2C slave address formats 71
C. FAQ - Frequent asked questions 73

© 2007-2010 Analytica GmbH

Chapter 1. The Programming interface of AnaGate devices

The *AnaGate* product line consist of several hardware devices, which offers access to different bus systems (I2C, SPI, CAN) or processors (Renesas) via standard network protocol.

The communication to the individual devices always is done through a documented and disclosed proprietary network protocol. Thus, all products which incorporates a socket interface (like personal computers, PLC, ...) are allowed to access the devices of the *AnaGate* product line.

Analytica provides a programming interface for users of Windows and Linux operating systems (X86) which implements the proprietary communication protocol and make it available through simple function calls. The software API (Application Programming Interface) is available free of charge for Windows and Linux operarting systems.

Table 1.1. Library files for Windows

Device	Windows library
AnaGate CAN	AnaGateCAN.dll
AnaGate CAN uno / duo / quattro	AnaGateCAN.dll
AnaGate CAN USB	AnaGateCAN.dll
AnaGate SPI	AnaGateSPI.dll
AnaGate I2C	AnaGateI2C.dll
AnaGate Universal Programmer	AnaGateSPI.dll, AnaGateI2C.dll

Note

To provide a widespread support of differrent programming languages like C++, Visual Basic, Delphi and the programming languages of the .NET family, the cdecl calling convention is used in all function calls. Using this calling convention means that all function parameters are pushed on the stack in reverse order (from right to left) and that the caller is responsible for the stack handling. Most programming languages support this calling convention.

Table 1.2. Library files for Linux

Device	Linux library (X86)	ARM9
AnaGate CAN	libCANdll.a, libAnaCommon.a	-
AnaGate CAN uno / duo / quattro	libCANdll.a, libAnaCommon.a	available
AnaGate CAN USB	libCANdll.a, libAnaCommon.a	available
AnaGate SPI	libSPIdll.a, libAnaCommon.a	-
AnaGate I2C	libI2Cdll.a, libAnaCommon.a	-
AnaGate Universal Programmer	libSPIdll.a, libI2Cdll.a, libAnaCommon.a	available

The different libraries include common and specific functions which are neccessary for accessing and controlling the devices of the *AnaGate* product line. In the following, all library functions of the software API are documented in detail.

© 2007-2010 Analytica GmbH

Tip

It is possible to extend individually the newer device models with embedded Linux (kernel 2.6) and ARM9 processor. The complete software API is available in a cross-compiled version and can be used on the devices itself to create individual device extensions. To do so is very easy because the programming interface on the personal computer and the device is completely identical.

A preconfigured virtual machine (Virtual-Box-Image) with Ubuntu-Linux “READY-to-USE” with installed development environment (**Kdevelop**, **Eclipse**) and all neccessary program libraries (**GCC**, cross compiler, libraries, **LUA**, ...) is available optional .

© 2007-2010 Analytica GmbH

Chapter 2. Notes concerning the communication protocol TCP

Access to the different models of the *AnaGate* product line is always done via the most frequently used network protocol TCP (Transmission Control Protocol).

TCP is connection-oriented packet-switched transport protocol which is located in layer 4 of the of the OSI reference model. In princible TCP is an end-to-end connection which allows exchange of data in both directions at the same time. An end-point is a pair formed of an IP address and a port number and. Such a pair builds a bidirectional software interface and is called socket.

The *AnaGate* device offers its functionality as so-called TCP server. It creates a socket with its IP address and a device-specific port number. On the models with CAN interface(s) a seperate socket with different port number is created for each existing CAN interface, on every socket up to 5 concurrent client connections are accepted. The SPI, I2C and Renesas interfaces accept only one concurrent connection at the same time.

Table 2.1. AnaGate devices and related port numbers

Device	Port number
AnaGate I2C, AnaGate Universal Programmer	5000
AnaGate CAN, AnaGate CAN uno	5001
AnaGate CAN duo	5001, 5101
AnaGate CAN quattro	5001, 5101, 5201, 5301
AnaGate SPI, AnaGate Universal Programmer	5002
AnaGate Renesas, AnaGate Universal Programmer	5008

Important

Please ensure that all used ports are set active on the personal computer to grant access to the *AnaGate* device. Any existing firewalls are to be configured accordingly.

2.1. Important properties of the network protocol

In most cases TCP is based on the internet protocol (IP). IP is package-oriented, whereby it is possible that data packets are lost or the packets can be received in wrong order or perhaps more than once.

TCP eliminates this behaviour and ensures that the the data packets are received in correct order at the recipient. Is a sent data packet not confirmed by the recipient within a timeout limit, the packet is sent again. Double packets are recognized at the recipient and are deleted. During connection the data transmission may be impaired, delayed or completely interrupted. A successful connection do not guarantee a permanently stable data transmission.

Detection and evaluation of network and line malfunctions can be difficult, if there is only sporadic communication on the line. How is possible to distinguish between a malfunction on the line or simply no data from the connected endpoint?

© 2007-2010 Analytica GmbH

To amend this problematic nature TCP provides an internal keep alive mechanism. Keep-alives are special data packets which are sent in regular intervalls between the two endpoints of an opened communication channel. The recipient of a keep-alive packet has to confirm the receipt to the sender within a certain period of time. Are there no keep-alives or confirmations of keep-alives receivedm the communication partner assumes that the channel is interrupted or the corresponding socket is malfunctioning.

The keep-alive mechanism of TCP is not active per default and has to be activated by the `setsockopt` function for each connection. The API functions which establish a connection to an *AnaGate* - like the `CANOpenDevice()` function - strictly activate the keep-alive mechisam of TCP.

Note

On Windows operating systems some settings concerning keep-alives can be set individually. These settings are valid for all network connection on this computer and can not be set individually for dedicated connections.

To do so the Windows registry keys **KeepAliveTime** and **KeepAliveInterval** of node `\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Tcpip\Parameters` has to be adjusted (administrator rights).

Especially the CAN-Ethernet gateways can be affected by the above described problems, for example if customer-specific needs ask for faster detection of connection aborts as possible via the standard mechanism. So, in the AnaGate models with CAN interface and linux OS an application-specific keep-alive algorithm is integrated in the device firmware to enhance connection control. On base of a predefined time period additional data packets are exchanged between the AnaGate hardware and the controlling unit/personal computer, which have to be confirmed by the corresponding endpoint (ALIVE_REQ, [TCP2010]). This integrated alive machanism can be activated individually on each connection with a different timeout interval.

Note

Users of the AnaGate-API do not have to implement the application-specific alive mechanism to use it. With a simple call to the API function `CANStartAlive` a concurrent thread is started which automatically monitors the communication channel time-controlled.

© 2007-2010 Analytica GmbH

Chapter 3. Common function reference

© 2007-2010 Analytica GmbH

DLLInfo

DLLInfo — Determines the current version information of the AnaGate DLL.

Syntax

#include <AnaGateDLL.h>

```
int DLLVersion(char * pcMessage, int nMessageLen);
```

Parameter

pcMessage Data buffer that is to accept the version reference number of the AnaGate DLL.

nMessageLen Size in bytes of the transferred data buffer.

Return value

Actual size of the returned version reference number.

Remarks

If the version reference number is too large for the transferred data buffer, it is abbreviated to the given number of characters (*nMessageLen*).

© 2007-2010 Analytica GmbH

Chapter 4. CAN API reference

The CAN API can be used with all CAN gateway models of the AnaGate series. The programming interface is identical for all devices and uses the network protocol TCP or UDP in general.

Following devices can be addresse via the CAN API interface:

- AnaGate CAN
- AnaGate CAN uno
- AnaGate CAN duo
- AnaGate CAN quattro
- AnaGate CAN USB

© 2007-2010 Analytica GmbH

CANOpenDevice, CANOpenDeviceEx

CANOpenDevice, CANOpenDeviceEx — Opens an network connection (TCP or UDP) to an AnaGate CAN device.

Syntax

#include <AnaGateDllCan.h>

```
int CANOpenDevice(int *pHandle, BOOL bSendDataConfirm, BOOL bSendDataInd, int nCANPort, const char * pcIPAddress, int nTimeout );
```

```
int CANOpenDeviceEx(int *pHandle, BOOL bSendDataConfirm, BOOL bSendDataInd, int nCANPort, const char * pcIPAddress, int nTimeout , int nSocketType );
```

Parameter

pHandle — Pointer to a variable, in which the access handle is saved in the event of a successful connection to the AnaGate device.

bSendDataConfirm — It set to TRUE, all incoming and outgoing Data requests are confirmed by the internal message protocol. Without confirmations a better transmittion performance is reached.

bSendDataInd — If set to FALSE, all incoming telegrams are discarded.

nCANPort — CAN port number. Allowed values are:

- `0` for port A (Modells AnaGate CAN uno, AnaGate CAN duo, AnaGate CAN quattro, AnaGate CAN USB and AnaGate CAN)
- `1` for port B (AnaGate CAN duo, AnaGate CAN quattro)
- `2` for port C (AnaGate CAN quattro)
- `3` for port D (AnaGate CAN quattro)

pcIPAddress — Network address of the AnaGate partner.

nTimeout — Default timeout for accessing the AnaGate in milliseconds.

A timeout is reported if the AnaGate partner does not respond within the defined timeout period. This global timeout value is valid on the current network connection for all commands and functions which do not offer a specific timeout value.

nSocketType — Specifies the socket type (ethernet layer 4) which is to be used for the new connection. Only two different types are supported: TCP and UDP. The funttion `CANOpenDevice` alyways uses TCP sockets. Use the following constants fpr the parameter:

`1` (SOCK_STREAM) TCP (Transmission Control Protocol)

`2` (SOCK_DGRAM) UDP (User Datagram Protocol)

 © 2007-2010 Analytica GmbH

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

Opens a TCP/IP connection to an CAN interface of a AnaGate CAN device. With `CANOpenDeviceEx` it is possible to set the ethernet layer4 protocol (tcp or udp). If the connection is established, CAN telegrams can be sent and received.

The connection should be closed with the function `CANCloseDevice` if not longer needed.

Important

It is recommended to close the connection, because all internally allocated system resources are freed again and the connected AnaGate device is signalled that the active connection is not longer in use and can be used again for new connect requests.

See the following example for the initial programming steps.

```
#include <AnaGateCANDll.h>
int main()
{
   int hHandle;
   int nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // ... now do something
       CANCloseDevice(hHandle);
   }
   return 0;
}
```

Remarks

The *`CANOpenDeviceEx`* function is supported for library versions 1.5-1.10 or higher and firmware version 1.3.7 or higher.

Device models of type AnaGate CAN (hardware version 1.1.A) do not listen for UPD connection requests. If trying to connect such a device via UPD, the `CANOpenDeviceEx` returns with a timeout error.

See also

CANCloseDevice

CANRestart

© 2007-2010 Analytica GmbH

CANCloseDevice

CANCloseDevice — Closes an open network connection to an AnaGate CAN device.

Syntax

#include <AnaGateDllCan.h>

```
int CANCloseDevice(int hHandle);
```

Parameter

hHandle Valid access handle.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

Closes an open network connection to an AnaGate CAN device. The *`hHandle`* parameter is a return value of a succesfull call to the function `CANOpenDevice`.

Important

It is recommended to close the connection, because all internally allocated system resources are freed again and the connected AnaGate device is signalled that the active connection is not longer in use and can be used again for new connect requests.

See also

CANOpenDevice, CANOpenDeviceEx

© 2007-2010 Analytica GmbH

CANSetGlobals

CANSetGlobals — Sets the global settings, which are to be used on the CAN bus

Syntax

#include <AnaGateDllCAN.h>

```
int CANSetGlobals(int hHandle, int nBaudrate, unsigned char nOperatingMode, BOOL bTermination, BOOL bHighSpeedMode, BOOL bTimeStampOn);
```

Parameter

hHandle Valid access handle.

nBaudrate The baud rate to be used. Following values are allowed:

- 10.000 für 10kBit
- 20.000 für 20kBit
- 50.000 für 50kBit
- 62.500 für 62,5kBit
- 100.000 für 100kBit
- 125.000 für 125kBit
- 250.000 für 250kBit
- 500.000 für 500kBit
- 800.000 für 800kBit (not AnaGate CAN)
- 1.000.000 für 1MBit

nOperatingMode The operating mode to be used. Following values are allowed.

- 0 = default mode.
- 1 = loop back mode: All telegrams sent by a connected partner is routed back to all active connected partners.
- 2 = listen mode: Device operates as passive bus partner, this means no telegrams are sent to the CAN bus (no ACKs for incoming telegrams too).

bTermination Use integrated CAN bus termination (TRUE= yes, FALSE = no). This setting is not supported by all AnaGate CAN models.

bHighSpeedMode Use high speed mode (TRUE= yes, FALSE= no). This setting is not supported by all AnaGate CAN models.

The high speed mode was created for large baud rates with continuously high bus load. In this mode telegrams are not confirmed on procol layer and the software filters defined via `CANSetFilter` are ignored.

 © 2007-2010 Analytica GmbH

bTimeStampOn — Use time stamp mode (TRUE= yes, FALSE= no). This setting is not supported by all AnaGate CAN models.

In activated time stamp mode an additional timestamp is sent with the CAN telegram. This timestamp indicates when the incomming message is received by the CAN controller or when the outgoing message is confirmed by the CAN controller.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

Sets the global settings of the used CAN interface. These settings are effective for all concurrent connections to the CAN interface. The settings are not saved permantently on the device and are reset every device restart.

Remarks

The settings of the integrated CAN bus termination, the high speed mode and the time stamp are not supported by the AnaGate CAN (hardware version 1.1.A). These settings are ignored by the device.

See also

CANGetGlobals

© 2007-2010 Analytica GmbH

CANGetGlobals

CANGetGlobals — Returns the currently used global settings on the CAN bus.

Syntax

#include <AnaGateDllCAN.h>

```
int CANGetGlobals(int hHandle, int * pnBaudrate, unsigned char * pnOperatingMode, BOOL * pbTermination, BOOL * pbHighSpeedMode, BOOL * pbTimeStampOn);
```

Parameter

hHandle Valid access handle.

pnBaudrate The baud rate currently used on the CAN bus.

pnOperatingMode The operating mode to be used. Following values are returned.

- 0 = default mode.
- 1 = loop back mode: All telegrams sent by a connected partner is routed back to all active connected partners.
- 2 = listen mode: Device operates as passive bus partner, this means no telegrams are sent to the CAN bus (no ACKs for incoming telegrams too).

pbTermination Is the integrated CAN bus termination used? (TRUE= yes, FALSE= no). This setting is not supported by all AnaGate CAn modells.

pbHighSpeedMode Is the high speed mode switched on? (TRUE= yes, = no). This setting is not supported by all AnaGate CAN modells.

The high speed mode was created for large baud rates with continuously high bus load. In this mode telegrams are not confirmed on procol layer and the software filters defined via `CANSetFilter` are ignored.

pbTimeStampOn Is a timestamp mode activated on the current network connection? (TRUE= yes, FALSE= no). This setting is not supported by all AnaGate CAN modells.

In activated time stamp mode an additional timestamp is sent with the CAN telegram. This timestamp indicates when the incomming message is received by the CAN controller or when the outgoing message is confirmed by the CAN controller.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

Returns the global settings of the used CAN interface. These settings are effective for all concurrent connections to the CAN interface.

 © 2007-2010 Analytica GmbH

Remarks

The settings of the integrated CAN bus termination, the high speed mode and the time stamp are not supported by the AnaGate CAN (hardware version 1.1.A). These settings are ignored by the device.

See also

CANGetGlobals

© 2007-2010 Analytica GmbH

CANSetFilter

CANSetFilter — Sets the current filter settings for the connection.

Syntax

#include <AnaGateDllCAN.h>

```
int CANSetFilter(int hHandle, const int * pnFilter);
```

Parameter

hHandle Valid access handle.

pnFilter Pointer to an array of 8 filter entries (4 mask and 4 range filter entries). A filter entry contains of two 32-bit values. Unused mask filter entries must be initialized with 0 values. Unused range filter entries must be initialized with a 0 for the start value and 0x1FFFFFFF for the end value.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

This function sets the current filter settings for the current connection. Filter can be used to suppress messages with specific CAN message ids.

A mask filter contains of a mask value, which defines the bits of the CAN identifier to examine, and the appropriate filter value. If the CAN identifier matches in the indicated filter mask with the filter value, the incoming CAN telegram is sent to the PC, otherwise not.

A range filter defines an address range with a appropriate start and end address. If the CAN identifier do not lie in the indicated filter range, the incoming CAN telegram is not sent to the PC.

Filter are only active, if the parameter *`bMonitor`* is set via the `CANOpenDevice` function.

See also

CANGetFilter

© 2007-2010 Analytica GmbH

CANGetFilter

CANGetFilter — Returns the current filter settings for the connection.

Syntax

#include <AnaGateDllCAN.h>

```
int CANGetFilter(int hHandle, int * pnFilter);
```

Parameter

hHandle Valid access handle.

pnFilter Pointer to an array of 8 filter entries (4 mask and 4 range filter entries). A filter entry contains of two 32-bit values. Unused mask filter entries are initialized with 0 values. Unused range filter entries are initialized with (0,0x1FFFFFFF) value pairs.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

This function retrieves the current filter settings for the current connection. Filter can be used to suppress messages with specific CAN message ids.

See also

CANSetFilter

© 2007-2010 Analytica GmbH

CANSetTime

CANSetTime — Sets the current system time on the AnaGate device.

Syntax

#include <AnaGateDllCAN.h>

```
int CANSetTime(int hHandle, long nSeconds, long nMicroseconds);
```

Parameter

hHandle Valid access handle.

nSeconds Time in seconds from 01.01.1970.

nMicroseconds Micro seconds.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

The `CANSetTime` function sets the system time on the AnaGate hardware.

If the time stamp mode is switched on by the `CANSetGlobals` function, the AnaGate hardware adds a time stamp to each incoming CAN telegram and a time stamp to the confirmation of a telegram sent via the API (only if confirmations are switched on for data requests).

Remarks

The `CANSetTime` function is supported by library version 1.4-1.8 or higher.

The setting of the base time for the time stamp mode is not supported by the AnaGate CAN (hardware version 1.1.A). This setting is ignored by the device.

© 2007-2010 Analytica GmbH

CANWrite, CANWriteEx

CANWrite, CANWriteEx — Send a CAN telegram to the CAN bus via the AnaGate device.

Syntax

#include <AnaGateDllCan.h>

```
int CANWrite(int hHandle, int nIdentifier, const char * pcBuffer, int nBufferLen, int nFlags);
```

```
int CANWriteEx(int hHandle, int nIdentifier, const char * pcBuffer, int nBufferLen, int nFlags, long * pnSeconds, long * pnMicroSeconds);
```

Parameter

hHandle	Valid access handle.
nIdentifier	CAN identifier of the sender. Parameter *nFlags* defines, if the address is in extended format (29-bit) or standard format (11-bit).
pcBuffer	Pointer to the data buffer.
nBufferLen	Length of data buffer (max. 8 bytes).
nFlags	The format flags are defined as follows. • Bit 0: If set, the CAN identifier is in extended format (29 bit), otherwise not (11 bit). • Bit 1: If set, the telegram is marked as remote frame.
pnSeconds	Timestamp of the confirmation of the CAN controller (seconds from 01.01.1970).
pnMicroSeconds	Micro seconds portion of the timestamp.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

Both functions sends a CAN telegram to the CAN bus via the AnaGate device.

The `CANWriteEx` additionallly returns a timestamp of the time at which the telegram is sent.

Note

With remote frames (RTR = remote transmission request) a destination node can request data from a source node. The data length is to be set to the number of requested bytes - on the CAN bus no data is sent only the data size information.

When using the `CANWrite` bzw. `CANWriteEx` functions to send remote frames the data buffer and the buffer size equal to the number of requested bytes have to be set correctly.

 © 2007-2010 Analytica GmbH

See the following example for sending a data telegram to the connected CAN bus.

```
#include <AnaGateCANDll.h>
int main()
{
   char cMsg[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
   int hHandle = 0;
   int nRC = 0;
   int nFlags = 0x0; //  11bit address + standard (not remote frame)
   int nIdentifier = 0x25; // send with CAN ID 0x25;

   int nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // send 8 bytes with CAN id 37
       nRC = CANWrite( hHandle, nIdentifier, cMsg, 8, nFlags );

       // send a remote frame to CAN id 37 (request 4 data bytes)
       nRC = CANWrite( hHandle, nIdentifier, cMsg, 4, 0x02 );

       CANCloseDevice(hHandle);
   }
   return 0;
}
```

Remarks

The `CANWriteEx` function is supported by library version 1.4-1.8 or higher.

For devices of type AnaGate CAN (hardware version 1.1.A) the function `CANWriteEx` is equal to `CANWrite`, the return values *`pnSeconds`* and *`pnMicroSeconds`* will remain unchanged.

© 2007-2010 Analytica GmbH

CANSetCallback, CANSetCallbackEx

CANSetCallback, CANSetCallbackEx — Defines an asynchronous callback function, which is called for each incoming CAN telegram.

Syntax

#include <AnaGateDllCan.h>

```
typedef void (WINAPI * CAN_PF_CALLBACK)(int nIdentifier, const char *
pcBuffer, int nBufferLen, int nFlags, int hHandle);

int CANSetCallback(int hHandle, CAN_PF_CALLBACK pCallbackFunction);

typedef void (WINAPI * CAN_PF_CALLBACK_EX)(int nIdentifier, const char
* pcBuffer, int nBufferLen, int nFlags, int hHandle);

int       CANSetCallbackEx(int       hHandle,       CAN_PF_CALLBACK_Ex
pCallbackFunctionEx);
```

Parameter

hHandle	Valid access handle.
pCallbackFunction	Function pointer to the private callback function. Set this parameter to `NULL` to deactivate the callback function. The parameters of the callback function are described in the documentation of the `CANWrite` funciton.
pCallbackFunctionEx	Function pointer to the private callback function. Set this parameter to `NULL` to deactivate the callback function. The parameters of the callback function are described in the documentation of the `CANWriteEx` funciton.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

Incoming CAN telegrams can bei received via a callback function, which can be set by a simple API call. If a callback function us set, is will be called by the API asynchronous.

Caution

The callback fucntion is called from a thread which is started by the CAN API and which is reading data from the socket. Because of this behaviour the callback code is executed by the thread context of the API and therefore it uses it the heap memeory of the API DLL and not the application program. So programming code should not use functions like `new`, `delete`, `alloc` or `free` which allocate, free or reallocate heap memory inside the callback.

See the following example for using a callback.

```
#include <AnaGateCANDll.h>

// Defintion of a callback, which writes incoming CAN data with timestamp to console
```

 © 2007-2010 Analytica GmbH

```
void WINAPI MyCallbackEx(int nIdentifier, const char * pcBuffer, int nBufferLen, int nFlags,
                         int hHandle, long nSeconds, long nMicroseconds)
{
   std::cout << "CAN-ID=" << nIdentifier << ", Data=";
   for ( int i = 0; i < nBufferLen; i++ )
   {
      std::cout << " 0x" << std::hex << int((unsigned char)(pcBuffer[i]));
   }
   time_t tTime = nSeconds;
   struct tm * psLocalTime = localtime(&tTime );
   std::cout << " " << std::setw(19) << asctime( psLocalTime ) << " ms(" << std::dec
             << std::setw(3) << nMicroseconds/1000 << "." << nMicroseconds%1000 << ")" << std::endl;
}

int main()
{
   int hHandle = 0;
   int nRC = 0;

   int nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // deactivate callback
       nRC = CANSetCallbackEx( hHandle, MyCallbackEx );

       getch();  // wait for keyboard input

       // deactivate callback
       nRC = CANSetCallbackEx( hHandle, 0 );

       CANCloseDevice(hHandle);
   }
   return 0;
}
```

Remarks

The two different callback functions have to be used depending on the active setting of the global timestamp option (`CANSetGlobals`). Only one of the callbacks can be activated at the same time.

See also

CANWrite, CANWriteEx

© 2007-2010 Analytica GmbH

CANReadDigital

CANReadDigital — Reads the current values of digital input and output registers of the AnaGate device.

Syntax

#include <AnaGateDllCan.h>

```
int CANReadDigital(int hHandle, unsigned long * pnInputBits, unsigned long * pnOutputBits);
```

Parameter

hHandle	Valid access handle.
pnInputBits	Pointer to the current value of the digital input register. Currently only bits 0 to 3 are used, other bits are reserved for future use and are set to 0.
pnOutputBits	Pointer to the current value of the digital output register. Currently only bits 0 to 3 are used, other bits are reserved for future use and are set to 0.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

All models of the AnaGate series (except the model AnaGate CAN uno in DIN rail case) have connectors for 4 digtial inputs and 4 digital outputs at the rear panel.

The current values of the digital inputs and outputs can be retrieved with the `CANReadDigital` function.

See the following example for setting an reading the digital IO.

```
#include <AnaGateCANDll.h>
int main()
{
   int hHandle = 0;
   int nRC = 0;
   unsigned long nInputs;
   unsigned long nOutputs = 0x03;

   int nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // set the digital output register (PIN 0 and PIN 1 to HIGH value)
       nRC = CANWriteDigital( hHandle, nOutputs );

       // read all input and output registers
       nRC = CANReadDigital( hHandle, &nInputs, &nOutputs );

       CANCloseDevice(hHandle);
   }
   return 0;
```

 © 2007-2010 Analytica GmbH

```
}
```

See also

CANWriteDigital

© 2007-2010 Analytica GmbH

CANWriteDigital

CANWriteDigital — Writes a new value to the digital output register of the AnaGate device.

Syntax

#include <AnaGateDllCAN.h>

```
int CANWriteDigital(int hHandle, unsigned long nOutputBits);
```

Parameter

hHandle	Valid access handle.
nOutputBits	New register value. Currently only bits 0 to 3 are used, other bits are reserved for future use.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

All models of the AnaGate series (except the model AnaGate CAN uno in DIN rail case) have connectors for 4 digtial inputs and 4 digital outputs ant the rear panel.

The digital outputs can be written with the `CANWriteDigital` function.

A simple example for reading/writing of the IOs can be found at the description of `CANReadDigital`.

See also

CANReadDigital

© 2007-2010 Analytica GmbH

CANRestart

CANRestart — Restarts a AnaGate CAN device.

Syntax

#include <AnaGateDllCan.h>

```
int CANRestart(const char * pcIPAddress, int nTimeout );
```

Parameter

pcIPAddress	Network address of the AnaGate partner.
nTimeout	Default timeout for accessing the AnaGate in milliseconds. A timeout is reported if the AnaGate partner does not respond within the defined timeout period.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

Restarts the AnaGate CAN device at the specified network address. It disconnects implicitly all open network connections to all existing CAN interfaces. The Restart command is even possible if the maximum number of allowed connections is reached.

Important

It is recommended to use this command only in emergency cases, if there is a need to connect even if the maximum number of concurrent connections is reached.

See also

CANOpenDevice

© 2007-2010 Analytica GmbH

CANDeviceConnectState

CANDeviceConnectState — Retrieves the current network connection state of the current AnaGate connection.

Syntax

#include <AnaGateDllCAN.h>

```
int CANDeviceConnectState(int hHandle);
```

Parameter

hHandle Valid connection handle of a successful call to CANOpenDevice.

Return value

Returns the current network connection state. Following values are possible:

- `1 = DISCONNECTED`: The connection to the AnaGate is disconnected.
- `2 = CONNECTING`: The connection is connecting.
- `3 = CONNECTED` : The connection is established.
- `4 = DISCONNECTING`: The connection is disonnecting.
- `5 = NOT_INITIALIZED`: The network protocol is not successfully initialized.

Description

This function can be used to check if an already connected device is disconnected.

The detection period of a state change depends on the use of internal AnaGate-ALIVE mechanism. This ALIVE mechanism has to be switched on explicitly via `CANStartAlive` function. Once activated the connection state is periodically checked by the ALIVE mechanism.

Remarks

The `CANDeviceConnectState` function is supported by library version 1.4-1.10 or higher.

See also

CANStartAlive

 © 2007-2010 Analytica GmbH

CANStartAlive

CANStartAlive — Starts the ALIVE mechanism, which checks periodically the state of the network connection to the AnaGate hardware.

Syntax

#include <AnaGateDllCan.h>

```
int CANStartAlive(int hHandle, int nAliveTime );
```

Parameter

hHandle	Valid access handle.
nAliveTime	Default time out in seconds for the ALIVE mechanism.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

The AnaGate communication protocol [TCP2010] supports an application specific connection control which allows faster detection of broken connection lines.

The `CANStartAlive` function starts a concurrent process in the DLL in order to send defined alive telegrams (ALIVE_REQ) peridically (approx. every half of the given time out) to the Anagate device via the current network connection. Is the alive telegram not confirmed within the alive time the connection is marked as `disconnected` and the socket is closed if not already closed.

Use the `CANDeviceConnectState` function to check the current network connection state.

Remarks

The `CANStartAlive` function is supported by library version 1.4-1.10 or higher.

It requires firmware version 1.3.8 or higher installed on the hardware, devices of type AnaGate CAN (hardware version 1.1.A) does not support the application specific alive mechanism.

See also

CANDeviceConnectState

Section 2.1, “ Important properties of the network protocol”

 © 2007-2010 Analytica GmbH

CANErrorMessage

CANErrorMessage — Returns a description of the given error code as a text string.

Syntax

#include <AnaGateDllCAN.h>

```
int CANErrorMessage(int nRetCode, char * pcMessage, int nMessageLen);
```

Parameter

nRetCode Error code for which the error description is to be determined.

pcMessage Data buffer that is to accept the error description.

nMessageLen Size in bytes of the transferred data buffer.

Return value

Actual size of the returned description.

Description

Returns a textual description of the parsed error code (see Appendix A, *Return Codes*). If the destination buffer is not big enough to store the text, the text is shortened to the specified buffer size.

See the following example in C/C++ language.

```
int nRC;
char cBuffer[200];
int nRC;
//... call a API function here
CANErrorMessage(nRC, cBuffer, 200);
std::cout << "Fehler: " << cBuffer << std::endl;
```

 © 2007-2010 Analytica GmbH

Chapter 5. SPI API reference

The Serial Peripheral Interface (SPI) is a synchroneous data link standard named by Motorola which operates in full duplex mode. The SPI gateway models of the AnaGate series provides access to a SPI bus via a standard networking.

With the SPI API these SPI gateways can be easily controlled. The programming interface is identical for all devices and used the network protocol TCP in general.

Following devices can be addresse via the SPI API interface:

- AnaGate SPI
- AnaGate Universal Programmer

© 2007-2010 Analytica GmbH

SPIOpenDevice

SPIOpenDevice — Opens a network connection to an AnaGate SPI device.

Syntax

#include <AnaGateDllSPI.h>

```
int SPIOpenDevice(int * pHandle, const char * pcIPAddress, int nTimeout);
```

Parameter

pHandle	Pointer to a variable, in which the access handle is saved in the event of a successful connection to the AnaGate device.
pcIPAddress	Network address of the AnaGate partner.
nTimeout	Default timeout for accessing the AnaGate in milliseconds. A timeout is reported if the AnaGate partner does not respond within the defined timeout period. This global timeout value is valid on the current network connection for all commands and functions which do not offer a specific timeout value.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

Opens a TCP/IP connection to an AnaGate SPI (resp. AnaGate Universal Programmer). After the connection is established, access to the SPI bus is possible.

Note

The AnaGate SPI (resp. the SPI interface of an AnaGate Universal Programmer) does not allow more than one concurrent network connection. During an established network connection all new connections are refused.

See the following example for the initial programming steps.

```
#include <AnaGateDllSPI.h>
int main()
{
   int hHandle;
   int nRC = SPIOpenDevice(&hHandle, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // ... now do something
       SPICloseDevice(hHandle);
   }
   return 0;
}
```

 © 2007-2010 Analytica GmbH

See also

SPICloseDevice

© 2007-2010 Analytica GmbH

SPICloseDevice

SPICloseDevice — Closes an open network connection to an AnaGate SPI device.

Syntax

#include <AnaGateDllSPI.h>

```
int SPICloseDevice(int hHandle);
```

Parameter

hHandle Valid access handle.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

Closes an open network connection to an AnaGate SPI device. The *`hHandle`* parameter is a return value of a succesfull call to the function `SPIOpenDevice`.

Important

It is recommended to close the connection, because all internally allocated system resources are freed again and the connected AnaGate device is signalled that the active connection is not longer in use and can be used again for new connect requests.

See also

SPIOpenDevice

© 2007-2010 Analytica GmbH

SPISetGlobals

SPISetGlobals — Sets the global settings, which are to be used on the AnaGate SPI.

Syntax

#include <AnaGateDllSPI.h>

```
int SPISetGlobals(int hHandle, int nBaudrate, unsigned char nSigLevel,
unsigned char nAuxVoltage, unsigned char nClockMode);
```

Parameter

hHandle Valid access handle.

nBaudrate The baud rate to be used. The values can be set individually, like

- 500.000 for 500kBit
- 1.000.000 for 1MBit
- 5.000.000 for 5MBit

Note

The required baud rate can be different from the value actually used because of internal hardware restrictions (frequency of the oscillator). If it is not possible to adjust the baud rate exactly to the parsed value, the nearest smaller possible value is used instead.

nSigLevel The voltage level for SPI signals to be used. Following values are allowed:

- 0 = Outputs in High Impedance Modus (Standard mode).
- 1 = +5.0V for the signals.
- 2 = +3.3V for the signals.
- 3 = +2.5V for the signals.

nAuxVoltage The voltage level of the support voltage to be used. Following values are allowed:

- 0 = support voltage is +3.3V.
- 1 = support voltage is 2.5V.

nClockMode The phase and polarity of the clock signal. Following values are allowed:

- 0 = CPHA=0 and CPOL=0.
- 1 = CPHA=0 and CPOL=1.
- 2 = CPHA=1 and CPOL=0.
- 3 = CPHA=1 and CPOL=1.

© 2007-2010 Analytica GmbH

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

Sets the global settings of SPI interface of the AnaGate SPI or the AnaGate Universal Programmer. These settings are not saved permantently on the device and are reset every device restart.

See also

SPIGetGlobals

© 2007-2010 Analytica GmbH

SPIGetGlobals

SPIGetGlobals — Returns the currently used global settings of the AnaGate SPI.

Syntax

#include <AnaGateDllSPI.h>

```
int SPIGetGlobals(int hHandle, int * pnBaudrate, unsigned char *
pnSigLevel, unsigned char * pnAuxVoltage, unsigned char * pnClockMode);
```

Parameter

hHandle Valid access handle.

pnBaudrate The baud rate currently used on the SPI bus in kBit.

pnSigLevel The voltage level currently used by the AnaGate SPI. Following values are possible:

- 0 = Outputs in High Impedance Modus (Standard mode).
- 1 = +5.0V for the signals.
- 2 = +3.3V for the signals.
- 3 = +2.5V for the signals.

pnAuxVoltage The voltage level of the support voltage currently used by the AnaGate SPI. Following values are possible:

- 0 = support voltage is +3.3V.
- 1 = support voltage is 2.5V.

pnClockMode The phase and polarity of the colck signal currently used by the AnaGate SPI. Following values are possible:

- 0 = CPHA=0 and CPOL=0.
- 1 = CPHA=0 and CPOL=1.
- 2 = CPHA=1 and CPOL=0.
- 3 = CPHA=1 and CPOL=1.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

Returns the currently used global settings of SPI interface of the AnaGate SPI or the AnaGate Universal Programmer.

 © 2007-2010 Analytica GmbH

See also

SPISetGlobals

© 2007-2010 Analytica GmbH

SPIDataReq

SPIDataReq — Writes and reads data to/from SPI bus.

Syntax

#include <AnaGateDllSPI.h>

```
int SPIDataReq(int hHandle, const char * pcBufWrite, int nBufWriteLen,
char * pcBufRead, int nBufReadLen);
```

Parameter

hHandle Valid access handle.

pcBufWrite Buffer with the data that is to be sent to the SPI partner.

nBufWriteLen Length of the data buffer *`pcBufWrite`* (byte count).

pcBufRead Byte buffer which holds the data received from the SPI partner.

nBufReadLen Number of bytes to read.

Description

Sends data to the SPI bus and receives data from the SPI bus.

On the SPI bus Data is transferred on two seperates data lines full duplex (SDO and SDI). The `SPIDatReq` has to split a single data transfer in two steps because of the spacial separation to the SPI bus. First the write data buffer is put into a TCP data telegram and sent to the AnaGate SPI. The AnaGate SPI makes the real data transfer on the SPI bus and send back a confirmation including the data received from the bus.

Important

It is impossible to detect that no device is present at the SPI bus. So, if no device is attached, the requested number of bytes are returned anyway - in this case the read buffer is filled with `0`.

See the following example for sending a command to the connected SPI bus.

```
#include <AnaGateDllSPI.h>
int main()
{
   char cBufWrite[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
   char cBufReceive[100];
   int hHandle = 0;
   int nRC = 0;

   int nRC = SPIOpenDevice(&hHandle, "192.168.1.254", 5000);
   if ( nRC == 0 )
   {
       // send 1 byte and receive 1 byte
       nRC = SPIDataReq( hHandle, cBufWrite, 1, cBufReceive, 1 );
       // send 1 byte and receive 5 byte
```

 © 2007-2010 Analytica GmbH

```
        nRC = SPIDataReq( hHandle, cBufWrite, 1, cBufReceive, 5 );
        // send 2 byte and receive 1 byte
        nRC = SPIDataReq( hHandle, cBufW2ite, 2, cBufReceive, 1 );

        SPICloseDevice(hHandle);
    }
    return 0;
}
```

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

© 2007-2010 Analytica GmbH

SPIReadDigital

SPIReadDigital — Reads the current values of digital input and output registers of the AnaGate device.

Syntax

#include <AnaGateDllSPI.h>

```
int SPIReadDigital(int hHandle, unsigned long * pnInputBits, unsigned long * pnOutputBits);
```

Parameter

hHandle	Valid access handle.
pnInputBits	Pointer to the current value of the digital input register. Currently only bits 0 to 3 are used, other bits are reserved for future use and are set to 0.
pnOutputBits	Pointer to the current value of the digital output register. Currently only bits 0 to 3 are used, other bits are reserved for future use and are set to 0.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

All models of the AnaGate series (except the model AnaGate CAN uno in DIN rail case) have connectors for 4 digtial inputs and 4 digital outputs at the rear panel.

The current values of the digital inputs and outputs can be retrieved with the `SPIReadDigital` function.

See the following example for setting an reading the digital IO.

```
#include <AnaGateDllSPI.h>
int main()
{
   int hHandle = 0;
   int nRC = 0;
   unsigned long nInputs;
   unsigned long nOutputs = 0x03;

   int nRC = SPIOpenDevice(&hHandle, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // set the digital output register (PIN 0 and PIN 1 to HIGH value)
       nRC = SPIWriteDigital( hHandle, nOutputs );

       // read all input and output registers
       nRC = SPIReadDigital( hHandle, &nInputs, &nOutputs );

       SPICloseDevice(hHandle);
   }
   return 0;
```

 © 2007-2010 Analytica GmbH

```
}
```

See also

SPIWriteDigital

© 2007-2010 Analytica GmbH

SPIWriteDigital

SPIWriteDigital — Write a new value to the digital output register of the AnaGate device.

Syntax

#include <AnaGateDllSPI.h>

```
int SPIWriteDigital(int hHandle, unsigned long nOutputBits);
```

Parameter

hHandle	Valid access handle.
nOutputBits	New register value. Currently only bits 0 to 3 are used, other bits are reserved for future use.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

All models of the AnaGate series (except the model AnaGate CAN uno in DIN rail case) have connectors for 4 digtial inputs and 4 digital outputs at the rear panel.

The digital outputs can be written with the `SPIWriteDigital` function.

A simple example for reading/writing of the IOs can be found at the description of `SPIReadDigital`.

See also

SPIReadDigital

© 2007-2010 Analytica GmbH

SPIErrorMessage

SPIErrorMessage — Returns a description of the given error code as a text string.

Syntax

#include <AnaGateDllCAN.h>

```
int SPIErrorMessage(int nRetCode, char * pcMessage, int nMessageLen);
```

Parameter

nRetCode Error code for which the error description is to be determined.

pcMessage Data buffer that is to accept the error description.

nMessageLen Size in bytes of the transferred data buffer.

Return value

Actual size of the returned description.

Description

Returns a textual description of the parsed error code (see Appendix A, *Return Codes*). If the destination buffer is not big enough to store the text, the text is shortened to the specified buffer size.

See the following example in C/C++ language.

```
int nRC;
char cBuffer[200];
int nRC;
//... call a API function here
SPIErrorMessage(nRC, cBuffer, 200);
std::cout << "Fehler: " << cBuffer << std::endl;
```


© 2007-2010 Analytica GmbH

Chapter 6. I2C API reference

Philips Semiconductors (now NXP Semiconductors) has developed a simple bidirectional 2-wire bus for efficient inter-IC control. This bus is called the Inter-IC or I2C-bus. Only two bus lines are required: a serial data line (SDA) and a serial clock line (SCL). Serial, 8-bit oriented, bidirectional data transfers can be made at up to 100 kbit/s in the Standard-mode, up to 400 kbit/s in the Fast-mode, up to 1 Mbit/s in the Fast-mode Plus (Fm+), or up to 3.4 Mbit/s in the High-speed mode. [NXP-I2C].

The I2C gateway models of the AnaGate series provides access to a I2C bus via a standard networking. With the I2C API these I2C gateways can be easily controlled. The programming interface is identical for all devices and used the network protocol TCP in general.

Following devices can be addresse via the I2C API interface:

- AnaGate I2C
- AnaGate Universal Programmer

© 2007-2010 Analytica GmbH

I2COpenDevice

I2COpenDevice — Opens a network connection to an AnaGate I2C or an AnaGate Universal Programmer).

Syntax

#include <AnaGateDllI2C.h>

```
int I2COpenDevice(int * pHandle, unsigned int nBaudrate, const char *
pcIPAddress, int nTimeout);
```

Parameter

pHandle — Pointer to a variable, in which the access handle is saved in the event of a successful connection to the AnaGate device.

nBaudrate — Baud rate to be used for the I2C bus. Teh value can be set individually, like

- 100000 for 100kBit (standard mode)
- 400000 for 400kBit (fast mode)

Note

pcIPAddress — Network address of the AnaGate partner.

nTimeout — Default timeout for accessing the AnaGate in milliseconds.

A timeout is reported if the AnaGate partner does not respond within the defined timeout period. This global timeout value is valid on the current network connection for all commands and functions which do not offer a specific timeout value.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

Opens a TCP/IP connection to an AnaGate I2C (resp. AnaGate Universal Programmer). After the connection is established, access to the I2C bus is possible.

Note

The AnaGate I2C (resp. the I2C interface of an AnaGate Universal Programmer) does not allow more than one concurrent network connection. During an established network connection all new connections are refused.

See the following example for the initial programming steps.

```
#include <AnaGateDllI2C.h>
```


© 2007-2010 Analytica GmbH

```
int main()
{
   int hHandle;
   int nRC = I2COpenDevice(&hHandle, 100000, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // ... now do something
       I2CCloseDevice(hHandle);
   }
   return 0;
}
```

See also

I2CCloseDevice

© 2007-2010 Analytica GmbH

I2CCloseDevice

I2CCloseDevice — Closes an open network connection to an AnaGate I2C device.

Syntax

#include <AnaGateDllI2C.h>

```
int I2CCloseDevice(int hHandle);
```

Parameter

hHandle Valid access handle.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

Closes an open network connection to an AnaGate I2C device. The *`hHandle`* parameter is a return value of a succesfull call to the function `I2COpenDevice`.

Important

It is recommended to close the connection, because all internally allocated system resources are freed again and the connected AnaGate device is signalled that the active connection is not longer in use and can be used again for new connect requests.

See also

I2COpenDevice

© 2007-2010 Analytica GmbH

I2CReset

I2CReset — Resets the I2C Controller in an AnaGate I2C device.

Syntax

#include <AnaGateDllI2C.h>

```
int I2CReset(int hHandle);
```

Parameter

hHandle Valid access handle.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

Resets the I2C Controller in an AnaGate I2C device.

© 2007-2010 Analytica GmbH

I2CRead

I2CRead — Reads data from an I2C partner.

Syntax

#include <AnaGateDllI2C.h>

```
int I2CRead(int hHandle, unsigned short nSlaveAddress, const char *
pcBuffer, int nBufferLen);
```

Parameter

hHandle	Valid access handle.
nSlaveAddress	Slave address of the I2C partner. The slave address can represent a so-called 7-bit or 10-bit address. (siehe Appendix B, *I2C slave address formats*).
pcBuffer	Byte buffer in which the data received from the I2C partner is to be stored.
nBufferLen	Number of bytes to read.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

Reads data from an I2C partner. The user must ensure that the setup of the data buffer and the address of the I2C partner are correct.

The R/W bit of the slave address does not have to be explicitly set by the user.

See also

I2CWrite

 © 2007-2010 Analytica GmbH

I2CWrite

I2CWrite — Writes data to an I2C partner.

Syntax

#include <AnaGateDllI2C.h>

```
int I2CWrite(int hHandle, unsigned short nSlaveAddress, const char *
pcBuffer, int nBufferLen);
```

Parameter

hHandle	Valid access handle.
nSlaveAddress	Slave address of the I2C partner. The slave address can represent a so-called 7-bit or 10-bit address. (siehe Appendix B, *I2C slave address formats*).
pcBuffer	Byte buffer with the data that is to be sent to the I2C partner.
nBufferLen	Size of bytes to be read.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

Writes data to an I2C partner. The user must ensure that the setup of the data buffer and the address of the I2C partner are correct.

The R/W bit of the slave address does not have to be explicitly set by the user.

See also

I2CRead

© 2007-2010 Analytica GmbH

I2CSequence

I2CSequence — This command is used to write a sequence of write and read commands to an I2C device.

Syntax

#include <AnaGateDllI2C.h>

```
int I2CSequence(int hHandle, const char * pcWriteBuffer, int nNumberOfBytesToWrite, char * pcReadBuffer, int nNumberOfBytesToRead, int * pnNumberOfBytesRead, int * pnByteNumberLastError);
```

Parameter

hHandle — Valid access handle.

pcWriteBuffer — byte buffer, containing the commands which are to be sent to the AnaGate I2C. The single commands are stored sequential in this byte buffer.

A read command is defined as follows:

Structure of read command for I2CSequence

Read command	Description
2 bytes (LSB,MSB)	slave address in 7- or 10-bit format, the R/W bit must be set to explicitely to `1`.
2 bytes (LSB,MSB)	number of data bytes to be read from the I2C device. The successfully read data bytes are stored in the *`pcReadBuffer`* buffer.

A write command is defined as follows:

Structure write command for I2CSequence

Write command	Description
2 bytes (LSB,MSB)	slave address in 7- or 10-bit format, the R/W bit must be set to explicitely to `1`.
2 bytes (LSB,MSB)	number of data bytes to be written to the I2C device.
N bytes	data bytes.

nNumberOfBytesToWrite — byte size of the data to write

pcReadBuffer — byte buffer, in which the received data is to be stored. The received data from different commands are stored in the buffer sequential (first the data of command 1, then the data of command 2, ...).

nNumberOfBytesRead — byte size of the read buffer (must be big enough for all included read requests)

pnNumberOfBytesRead — byte count, which is read from I2C.

 © 2007-2010 Analytica GmbH

pnByteNumberLastError	Number of byte in the *`pcWriteBuffer`* buffer, which raises an error.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

The user must ensure that the setup of the data buffer and the address of the I2C partner are correct.

© 2007-2010 Analytica GmbH

I2CReadDigital

I2CReadDigital — Reads the current values of digital input and output registers of the AnaGate device.

Syntax

#include <AnaGateDllI2C.h>

```
int I2CReadDigital(int hHandle, unsigned long * pnInputBits, unsigned long * pnOutputBits);
```

Parameter

hHandle	Valid access handle.
pnInputBits	Pointer to the current value of the digital input register. Currently only bits 0 to 3 are used, other bits are reserved for future use and are set to 0.
pnOutputBits	Pointer to the current value of the digital output register. Currently only bits 0 to 3 are used, other bits are reserved for future use and are set to 0.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

All models of the AnaGate series (except the model AnaGate CAN uno in DIN rail case) have connectors for 4 digtial inputs and 4 digital outputs at the rear panel.

The current values of the digital inputs and outputs can be retrieved with the `I2CReadDigital` function.

See the following example for setting an reading the digital IO.

```
#include <AnaGateDllI2C.h>
int main()
{
   int hHandle = 0;
   int nRC = 0;
   unsigned long nInputs;
   unsigned long nOutputs = 0x03;

   int nRC = I2COpenDevice(&hHandle, 400000, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // set the digital output register (PIN 0 and PIN 1 to HIGH value)
       nRC = I2CWriteDigital( hHandle, nOutputs );

       // read all input and output registers
       nRC = I2CReadDigital( hHandle, &nInputs, &nOutputs );

       CANCloseDevice(hHandle);
   }
   return 0;
```

 © 2007-2010 Analytica GmbH

```
}
```

See also

I2CWriteDigital

© 2007-2010 Analytica GmbH

I2CWriteDigital

I2CWriteDigital — Writes a new value to the digital output register of the AnaGate device.

Syntax

#include <AnaGateDllCAN.h>

```
int I2CWriteDigital(int hHandle, unsigned long nOutputBits);
```

Parameter

hHandle	Valid access handle.
nOutputBits	New register value. Currently only bits 0 to 3 are used, other bits are reserved for future use.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

Description

All models of the AnaGate series (except the model AnaGate CAN uno in DIN rail case) have connectors for 4 digtial inputs and 4 digital outputs at the rear panel.

The digital outputs can be written with the `I2CWriteDigital` function.

A simple example for reading/writing of the IOs can be found at the description of `I2CReadDigital`.

See also

I2CReadDigital

© 2007-2010 Analytica GmbH

I2CErrorMessage

I2CErrorMessage — Returns a description of the given error code as a text string.

Syntax

#include <AnaGateDllCAN.h>

```
int I2CErrorMessage(int nRetCode, char * pcMessage, int nMessageLen);
```

Parameter

nRetCode Error code for which the error description is to be determined.

pcMessage Data buffer that is to accept the error description.

nMessageLen Size in bytes of the transferred data buffer.

Return value

Actual size of the returned description.

Description

Returns a textual description of the parsed error code (see Appendix A, *Return Codes*). If the destination buffer is not big enough to store the text, the text is shortened to the specified buffer size.

See the following example in C/C++ language.

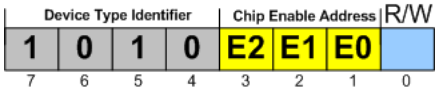
```
int nRC;
char cBuffer[200];
int nRC;
//... call a API function here
I2CErrorMessage(nRC, cBuffer, 200);
std::cout << "Fehler: " << cBuffer << std::endl;
```

 © 2007-2010 Analytica GmbH

6.1. Programming I2C EEPROM

The AnaGate I2C is very well suited for programming serial I2C EEPROM. To support this special requirement two different functions are made available: `I2CReadEEPROM` and `I2CWriteEEPROM`.

Like all other I2C-capable devices EEPROM's are addressable on the I2C bus via a unique slave address (see also Appendix B, *I2C slave address formats*). The so-called *Device Type Identifier* for these types of devcies is `0xA`. In principle 8 similar devices can be connected and addressed via the *Chip Enable Bits* E0, E1 und E0.



A data transmission is started with a **Start** signal by the master, followed by the slave address. The slave address is confirmed by the slave with a **ACK**. Depending on the R/W bit data is written (data to slave) or read (data from slave). The last byte of a read access has to be confirmed with a **NAK** by the master to signal the slave end of read transmission. The data transmission is terminated always by a **Stop** signal from the master.

When using EEPROM's the memory address is transmitted after transmission of the slave address, to advice the slave which memory address is to be written or read. Depending on the used EEPROM type the memory address is sent as single byte (8 bit) or as two bytes (16 bit, MSB First).

To expand the address space from 8 bit (or 16 bit), some EEPROM types use the *Chip Enable Bits* E0, E1, E2 as additional address bits. Which bits are used in individual cases is defined by the chip producer. In the following, all possible combinations of the bits usage are listed:

Table 6.1. Usage of the CHIP-Enable Bits of I2C EEPROMs

Mode [1]	**Usage**	**Description**
0x0	E2-E1-E0	Bits are only used to select the chip.
0x1	E2-E1-A0	Bit E0 is used to expand the addresse space. It is used for address bit A8 (resp. A16).
0x2	E2-A1-A0	E0 and E1 are used to expand the addresse space. E0 is used for address bit A8 (resp. A16) and E1 is used for A9 (resp. A17).
0x3	A2-A1-A0	E0, E1 and E2 are used to expand the addresse space. E0 is used for address bit A8 (resp. A16), E1 for A9 (resp. A17) and E2 for A10 (resp. A18).
0x5	A0-E1-E0	Bit E2 is used to expand the addresse space. It is used for address bit A8 (resp. A16). Das E2-Bit wird für die Addressierung verwendet. Es entspricht dabei dem Adressbit A8 bzw. A16.
0x6	A1-A0-E0	E2 and E1 are used to expand the addresse space. E1 is used for address bit A8 (resp. A16) and E2 is used for A9 (resp. A17).

[1] Set this mode flag in bit 8-10 of parameter *`nOffsetFormat`* in I2CReadEEPROM and I2CWriteEEPROM functions.

© 2007-2010 Analytica GmbH

I2CReadEEPROM

I2CReadEEPROM — Reads data from an EEPROM on the I2C bus.

Syntax

#include <AnaGateDllI2C.h>

```
int I2CReadEEPROM(int hHandle, unsigned short nSubAddress, unsigned int nOffset, const char * pcBuffer, int nBufferLen, unsigned int nOffsetFormat);
```

Parameter

hHandle	Valid access handle.
nSubAddress	Subaddress of the EEPROM to communicate with. The valid values for *nSubAddress* are governed by the setting used in the parameter *nOffsetFormat* (bits 8-10). If the EEPROM type needs to use bits of the *Chip Enable Address* to address the internal memory, only the remaining bits can be used to select the device itself. • No bit is used for addressing: 0 to 7 • 1 bit is used for addressing: 0 to 3 • 2 bits are used for addressing: 0 to 1 • 3 bits are used for addressing: 0
nOffset	Data offset on the EEPROM from which the transferred data is to be read.
pcBuffer	Character string buffer in which the received data is to be stored.
nBufferLen	Length of the data buffer.
nOffsetFormat	Defines how a memory address of EEPROM has to be specified when accessing the device. Bits 0-7 indicate the number of bits which are used in the address byte (word) of the I2C command for addressing the device memory. Bits 8-10 indicate how much and which bits of the *Chip Enable Bits* are used for addressing the device memory (see Table 6.1, “ Usage of the CHIP-Enable Bits of I2C EEPROMs” for allowed values).

Note

The maximum addressable size of an EEPROM is derived from the sum of all the bits. For example a M24C08 uses 8 bits of the address byte and an extra bit in the slave address. The total 9 bits can address up to 512 bytes.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

© 2007-2010 Analytica GmbH

Description

The `I2CReadEEPROM` function reads data from an I2C EEPROM.

Of course all access to the memory of an EEPROM is done by standard I2C read or write commands. So, when reading from the memory only the matching slave address, the memory offset address and the data has to be sent to the I2C bus.

`I2CReadEEPROM` translates the given memory address on the chip by means of the sub address and the addressing mode of the present EEPROM type. The slave address of the EEPROM is automatically determined and not mandatory for the fucntion call.

A programming example which clears a **ST24C1024** can be found at the description of I2WriteEEPROM.

See also

I2CWriteEEPROM

© 2007-2010 Analytica GmbH

I2CWriteEEPROM

I2CWriteEEPROM — Writes data to an I2C EEPROM.

Syntax

#include <AnaGateDllI2C.h>

```
int I2CWriteEEPROM(int hHandle, unsigned short nSubAddress, unsigned int nOffset, const char * pcBuffer, int nBufferLen, unsigned int nOffsetFormat);
```

Parameter

hHandle	Valid access handle.
nSubAddress	Subaddress of the EEPROM to communicate with. The valid values for *nSubAddress* are governed by the setting used in the parameter *nOffsetFormat* (bits 8-10). If the EEPROM type needs to use bits of the *Chip Enable Address* to address the internal memory, only the remaining bits can be used to select the device itself. • No bit is used for addressing: 0 to 7 • 1 bit is used for addressing: 0 to 3 • 2 bits are used for addressing: 0 to 1 • 3 bits are used for addressing: 0
nOffset	Data offset on the EEPROM to which the transferred data is to be written.
pcBuffer	Character string buffer with the data that is to be written.
nBufferLen	Length of the data buffer.
nOffsetFormat	Defines how a memory address of EEPROM has to be specified when accessing the device. Bits 0-7 indicate the number of bits which are used in the address byte (word) of the I2C command for addressing the device memory. Bits 8-10 indicate how much and which bits of the *Chip Enable Bits* are used for addressing the device memory (see Table 6.1, “ Usage of the CHIP-Enable Bits of I2C EEPROMs” for allowed values).

Note

The maximum addressable size of an EEPROM is derived from the sum of all the bits. For example a M24C08 uses 8 bits of the address byte and an extra bit in the slave address. The total 9 bits can address up to 512 bytes.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *Return Codes*).

 © 2007-2010 Analytica GmbH

Description

The `I2CWriteEEPROM` function writes data to an I2C EEPROM.

Of course all access to the memory of an EEPROM is done by standard I2C read or write commands. So, when writing to the memory only the matching slave address, the memory offset address and the data has to be sent to the I2C bus.

`I2CWriteEEPROM` translates the given memory address on the chip by means of the sub address and the addressing mode of the present EEPROM type. The slave address of the EEPROM is automatically determined and not mandatory for the fucntion call.

Tip

It is important to note that an EEPROM is divided into memory pages, and that a single write command can only program data within a page. Users of `I2CWriteEEPROM` must ensure to do not write across page limits. The page size depends on the EEEPOM type.

See the following example for writing data to a **ST24C1024**.

```
#include <AnaGateDllSPI.h>
int main()
{
   char           cBufferPage[256];
   int            hHandle = 0;
   int            nRC = 0;
   unsigned short nSubAddress = 0; 1
   unsigned int   nOffsetFormat = 0x10|0x0F; 2

   int nRC = I2COpenDevice(&hHandle, 400000, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       memset(cBufferPage,0,256);  // clear page buffer
       for (int i=0; i<512;i++)
       {
          I2CWriteEEPROM( hHandle, nSubAddress, i*256, cBufferPage, 256, nOffsetFormat ); 3
       }
       I2CCloseDevice(hHandle);
   }
   return 0;
}
```

1. It is possible to address 4 individual ST24C1024 on a single I2C bus. By selection of subaddress `0` the control pins E2 and E1 have to be LOW.
2. 17 address bits are used to address the 128KB of a ST24C1024. 16 bits are set via the address bytes of the write command: `16=0x0F`. The address bit A16 is set via the E0 bit of the *Chip Enable Address*, therefore addressing mode `1` (E2-E1-A0) must be set: `0x10`.
3. The page size of a ST24C1024 is 256 byte, every page is programmed full within the for-loop.

See also

I2CReadEEPROM

© 2007-2010 Analytica GmbH

Chapter 7. Programming examples

7.1. Programming language C/C++

The AnaGate programming API can be used on Windows systems as well as on linux systems (X86). All available API functions are coded operating system independant, so that source code once created can be used on both operating systems. Only the way the libraries are linked on the different operating systems or different compilers have to be customized.

Windows operating systems

There are basically two ways to access the API functions for the C/C++ programmer:

- When directly accessing the library all functions has to made known by preceding calls to the Win32 methods `LoadLibrary` and `GetProcAddress`.
- The functions can be alternatively accessed easily via an import library, which automatically loads all DLL functions and make them available implicitely.

In the following examples it is assumed that the second option is used and the corresponding import library is linked.

7.1.1. CAN Console application C/C++ (MSVC)

This programming example for C++ demonstrates how to connect to an AnaGate CAN and how to process received CAN data via a callback function.

Note

The source code of the example can be found in directory `Samples/CAN-VC6` resp. `Samples/CAN-VC7` on the CD.

```
#include "AnaGateDLLCAN.h"

WINAPI void MyCallback(int nIdentifier, const char * pcBuffer, int nBufLen, int nFlags, int nHandle)
{
    std::cout << "CAN-ID=" << nIdentifier << ", Data=";
    for ( int i = 0; i < nBufLen; i++ )
    {
       std::cout " 0x" << std::hex << pcBuffer[i];
    }
    std::cout << std::endl;
}

int main( )
{
   int hHandle = NULL;

   // opens AnaGate CAN duo device on port A, timeout after 1000 millseconds
   int nRC = CANOpenDevice(&hHandle,FALSE,TRUE,0,"192.168.2.1",1000);

   if ( nRC == 0 )
   {
      nRC = CANSetCallback(hHandle,MyCallback);
      getch();  // wait for keyborad input
   }
   if ( nRC == 0 )
   {
      nRC = CANCloseDevice(hHandle);// close device
   }
}
```


© 2007-2010 Analytica GmbH

7.2. Programming language Visual Basic 6

As already described in the previous chapters, the libraries of the *AnaGate*-API use the cdecl calling convention to parse function parameters to the program stack. Unfortunately this is generally not supported by the programming language *Visual Basic 6*.

To work around this limitiation, the libraries for the AnaGate devices are available in a specfic version for programming VB6 applications. In this versions the stdcall calling convention is used, which is the only supported by VB6. Except for the way the parameters are pushed on the stack these specific VB6 versions of the API libraries are exactly identical to the standard versions.

Note

Use library `AnaGateCAN.dll` instead of library `AnaGateCANVB6.dll`.

Use library `AnaGateSPI.dll` instead of library `AnaGateSPIVB6.dll`.

7.2.1. SPI Example with user interface for VB6

This programming example for Visual Basisc 6 demonstrates how to connect to an AnaGate SPI and how to execute a command on the SPI bus:

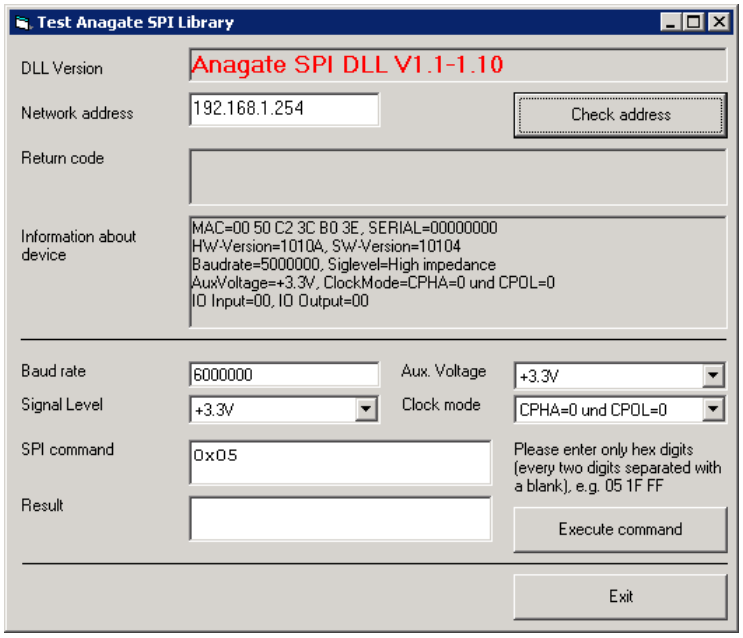
- Getting the global device settings like baud rate for example
- Sending a single command to the SPI bus

Note

The source code of the example can be found in directory `Samples/SPI-VB6` on the CD.

7.2.1.1. User interface

Figure 7.1. Input form of SPI example (VB6)



© 2007-2010 Analytica GmbH

Dialog fields

Network address	Network address of the AnaGate SPI.
Check address	Establishs a connection to the AnaGate SPI with the specified network address and reads back some device information and global device settings.
Baud rate	The baud rate to be used. The value can be set individually.
Signal Level	The voltage level for SPI signals to be used.
Aux. Voltage	The voltage level of the support voltage to be used.
Clock mode	The phase and polarity of the clock signal.
SPI command	SPI command to be sent to the connected SPI device. The command has to be entered as blank-separated hexadecimal byte groups ("`05 1F 3A`" for example).
Execute command	Executes a SPI command and displays the result in the *`Result`* dialog field. Please keep in mind, that the SPI bus is used as full duplex line, this means that data is written and received parallel. Make sure that you write the same number of bytes to the bus as you want to receive (in this cace add padding bytes to the SPI command). For example is the **Read Status Register** command of a **M25P80** defined as `0x05`. The result of the command is a single byte (8 bit) representing the current value of the Status Register.

7.2.1.2. Getting global device settings

All SPI related functions of the AnaGate API are declared for Visual Basic users in `AnaGateSPI.bas` and are read-to-use. The following code snipplet includes some exemplary declarations of the API functions used below.

```
Public Declare Function SPIOpenDevice Lib "AnaGateSPIVB6" _
               Alias "_SPIOpenDevice@12" (ByRef Handle As Long, _
                                          ByVal TCPAddress As String, _
                                          ByVal Timeout As Long) As Long

Public Declare Function SPICloseDevice Lib "AnaGateSPIVB6" _
               Alias "_SPICloseDevice@4" (ByVal Handle As Long) As Long

Public Declare Function SPIGetGlobals Lib "AnaGateSPIVB6" _
              Alias "_SPIGetGlobals@20" (ByVal hHandle As Long, _
                                         ByRef nBaudrate As Long, _
                                         ByRef SigLevel As Byte, _
                                         ByRef nAuxVoltage As Byte, _
                                         ByRef nClockMode As Byte) As Long
```

The event procedure `btnCheckAddress_Click` is called on click of the *Check Address* button.

```
Private Sub btnCheckAddress_Click()
    Dim nRC As Long, Data As String, sText As String, I As Long

    nRC = SPIOpenDevice(hHandle, Me.IPAddresse.Text, 2000) 1
    If nRC <> 0 Then
        Me.lblErrorMsg.Caption = "Fehler bei SPIOpenDevice: " & GetErrorMsg(nRC)
    Else
        Me.lblErrorMsg.Caption = GetAnagateInfo(hHandle)
    End If
    nRC = SPICloseDevice(hHandle) 2
End Sub
```


© 2007-2010 Analytica GmbH

1. A call to `SPIOpenDevice` establishes a network connection to the device. If the function fails, a textual error description is returned via Funktion `GetErrorMsg`.
2. The connection to the device is closed with the `SPICloseDevice` function.

Reading the device settings and creation of the textual presentation of the data is done by the `GetAnagateInfo` function.

```
Private Function GetAnagateInfo(hHandle As Long) As String
   Dim nRC As Long, sText As String
   Dim nBaudrate As Long, nSigLevel As Byte, nAuxVoltage As Byte, nClockMode As Byte
   Dim nDigitalOutput As Long, nDigitalInput As Long

   nRC = SPIGetGlobals(hHandle, nBaudrate, nSigLevel, nAuxVoltage, nClockMode)
   If (nRC = 0) Then
        sText = sText & "Baudrate=" & CStr(nBaudrate) & ", Siglevel="
        Select Case nSigLevel
            Case 1: sText = sText & "+5.0V"
            Case 2: sText = sText & "+3.3V"
            Case 3: sText = sText & "+2.5V"
            Case Else: sText = sText & "High impedance"
        End Select
        sText = sText & vbCrLf & "AuxVoltage="
        Select Case nAuxVoltage
            Case 1: sText = sText & "+2.5V"
            Case Else: sText = sText & "+3.3V"
        End Select
        sText = sText & ", ClockMode="
        Select Case nClockMode
            Case 1: sText = sText & "CPHA=0 und CPOL=1"
            Case 2: sText = sText & "CPHA=1 und CPOL=0"
            Case 3: sText = sText & "CPHA=1 und CPOL=1"
            Case Else: sText = sText & "CPHA=0 und CPOL=0"
        End Select
   Else
      sText = sText & "Fehler bei SPIGetGlobals: " & GetErrorMsg(nRC) & vbCrLf
   End If
   GetAnagateInfo = sText
End Function
```

7.2.1.3. Executing a command on the SPI bus

The AnaGate SPI can send arbitrary commands to the connected SPI bus. To write and read data by the PC only the `SPIDataReq` function is required.

```
Public Declare Function SPIDataReq Lib "AnaGateSPIVB6" Alias "_SPIDataReq@20" (ByVal hHandle As Long, _
                                    ByVal lpBufferWrite As Any, _
                                    ByVal nBufferWriteLen As Long, _
                                    ByVal lpBufferRead As Any, _
                                    ByVal nBufferReadlLen As Long) As Long
```

The event procedure `btnStart_Click` is called on click of the *Execute command* button.

```
Private Sub btnStart_Click()
    Dim nRC As Long, sText As String, I As Integer, sByteText As String
    Dim nBaudrate As Long, nSigLevel As Byte, nAuxVoltage As Byte, nClockMode As Byte
    Dim nBufferWriteLen As Long, nBufferReadLen As Long
    Dim arrWrite(1 To 255) As Byte, arrRead(1 To 255) As Byte

    nRC = SPIOpenDevice(hHandle, Me.IPAddresse.Text, 2000)
    If nRC <> 0 Then
        sText = "Fehler bei SPIOpenDevice: " & GetErrorMsg(nRC)
    Else
       nBaudrate = CLng(Me.txtBaudrate)
       nSigLevel = CLng(Me.cmbSigLevel.ListIndex)
       nAuxVoltage = CLng(Me.cmbAuxVoltage.ListIndex)
       nClockMode = CLng(Me.cmbClockMode.ListIndex)
       nRC = SPISetGlobals(hHandle, nBaudrate, nSigLevel, nAuxVoltage, nClockMode) 1

       If nRC <> 0 Then
             sText = sText & "Fehler bei SPISetGlobals: " & GetErrorMsg(nRC) & vbCrLf
```

 © 2007-2010 Analytica GmbH

```
       End If
       Me.lblDeviceInfo.Caption = GetAnagateInfo(hHandle)

       nBufferWriteLen = GetCommand(arrWrite)  2
       nBufferReadLen = nBufferWriteLen

       nRC = SPIDataReq(hHandle, VarPtr(arrWrite(1)), nBufferWriteLen, _
                                 VarPtr(arrRead(1)), nBufferReadLen)      3
       If nRC = 0 Then
           For I = 1 To nBufferReadLen
              sByteText = sByteText & "0x" & ToHex(arrRead(I)) & " "
           Next I
           Me.txtBufferRead = sByteText
           sText = sText & "SPIDataReq OK: " & vbCrLf
       Else
           sText = sText & "Fehler bei SPIDataReq: " & GetErrorMsg(nRC) & vbCrLf
       End If

       nRC = SPICloseDevice(hHandle)
    End If

End Sub
```

1. A call to `SPISetGlobals` sets the global settings on the device, the parameter values of the input fields in the dialog form are used.
2. The `GetCommand` function converts the texual SPI command entered in the input field on the form to a byte array structure.
3. To process the data in the read and receive buffers, a byte array is used as VB6 data type for both buffers. For this to work, the real memory address of the array data has to be parsed to the DLL function. This will be done by using the `VarPtr` function on the first byte array element.

.

7.3. Programming language VB.NET

Of course, it is also possible to use the functions of the AnaGate API with the .NET programming languages. For these languages the functions have only to be declared correctly in one of the .NET languages. Loading and unloading of the declared API functions is done automatically by the .NET framework.

7.3.1. CAN Console application VB.NET

This programming example for VB.NET demonstrates how to connect to an AnaGate CAN and how to process received CAN data via a callback function.

```
Declare Function CANOpenDevice Lib "AnaGateCAN" (ByRef Handle As Int32, _
                                                 ByVal ConfirmData As Int32, _
                                                 ByVal MonitorOn As Int32, _
                                                 ByVal PortNumber As Int32, _
                                                 ByVal TCPAddress As String, _
                                                 ByVal Timeout As Int32) As Int32
Declare Function CANCloseDevice Lib "AnaGateCAN" (ByVal Handle As Int32) As Int32
Public Delegate Sub CAN_CALLBACK(ByVal ID As Int32, ByVal Buffer As IntPtr, _
                                 ByVal Bufferlen As Int32, ByVal Flags as Int32, _
                                 ByVal Handle as Int32)
Declare Function CANSetCallback Lib "AnaGateCAN" (ByVal Handle As Int32, _
                                                  ByVal MyCB As CAN_CALLBACK) As Int32

Sub CANCallback( ByVal ID As Int32, ByVal Buffer As IntPtr, ByVal Bufferlen As Int32, _
                 ByVal Flags as Int32, ByVal Handle as Int32)
  Dim Bytes as Byte(8)

  System.Runtime.InteropServices.Marshal.Copy(Buffer, Bytes, 0, Bufferlen )
  Console.Out.Write( "CAN-ID=" )
  Console.Out.Write( ID )
```

 © 2007-2010 Analytica GmbH

```
  Console.Out.Write( ",Data=" )
  For I As Integer = 0 To BufferLen - 1
    Console.Out.Write( Bytes(I) )
  Next

End Sub

Function Main(ByVal CmdArgs() As String) As Integer

  'Opens the single CAN port of a AnaGate CAN
  dim RC as Int32 = CANOpenDevice(Handle, 0, 1, 400, 0, "192.168.2.1", 1000)
  If RC = 0 Then
    CANSetCallback( Handle, AddressOf CANCallback )
  end If
  If RC = 0 Then
    CANCloseDevice( Handle )
  end if
 end Function
```


© 2007-2010 Analytica GmbH

Appendix A. Return Codes

API return codes

Followed a list of the return values of the API functions. This values are defined in the header file `AnaGateErrors.h`.

Table A.1. Common return values for all devices of AnaGate series

Value	Name	Description
0	ERR_NONE	No errors.
0x000001	ERR_OPEN_MAX_CONN	Open failed, maximal count of connections reached.
0x0000FF	ERR_OP_CMD_FAILED	Command failed with unknown failure.
0x020000	ERR_TCPIP_SOCKET	Socket error in TCP/IP layer occured.
0x030000	ERR_TCPIP_NOTCONNECTED	Connection to TCP/IP partner can't established or is disconnected.
0x040000	ERR_TCPIP_TIMEOUT	No answer received from TCP/IP partner within the defined timeout.
0x050000	ERR_TCPIP_CALLNOTALLOWED	Command is not allowed at this time.
0x060000	ERR_TCPIP_NOT_INITIALIZED	TCP/IP-Stack can't be initialized.
0x0A0000	ERR_INVALID_CRC	AnaGate TCP/IP telegram has incorrect checksum (CRC).
0x0B0000	ERR_INVALID_CONF	AnaGate TCP/IP telegram wasn't receipted from partner.
0x0C0000	ERR_INVALID_CONF_DATA	AnaGate TCP/IP telegram wasn't receipted correct from partner.
0x900000	ERR_INVALID_DEVICE_HANDLE	Invalid device handle.
0x910000	ERR_INVALID_DEVICE_TYPE	Function can't be executed on this device handle, as she is assigned to another device type of AnaGate series.

Table A.2. Return values for AnaGate I2C

Value	Name	Description
0x000120	ERR_I2C_NACK	I2C-NACK
0x000121	ERR_I2C_TIMEOUT	I2C Timeout

A textual description of the return value can be retrieved with the function `I2CErrorMessage()`.

Table A.3. Return values for AnaGate CAN

Value	Name	Description
0x000220	ERR_CAN_NACK	CAN-NACK
0x000221	ERR_CAN_TX_ERROR	CAN Transmit Error
0x000222	ERR_CAN_TX_BUF_OVERLOW	CAN buffer overflow
0x000223	ERR_CAN_TX_MLOA	CAN Lost Arbitration
0x000224	ERR_CAN_NO_VALID_BAUDRATE	CAN Setting no valid Baudrate

A textual description of the return value can be retrieved with the function `CANErrorMessage()`.

© 2007-2010 Analytica GmbH

Table A.4. Return values for AnaGate Renesas

Value	Name	Description
0x000920	ERR_RENESAS_TIMEOUT	Renesas timeout
0x000921	ERR_RENESAS_INVALID_ID	Renesas Invalid ID
0x000922	ERR_RENESAS_FLASH_ERASE_FAILED	Renesas failed erase the flash
0x000923	ERR_RENESAS_PAGE_PROG_FAILED	Renesas failed prog the page

A textual description of the return value can be retrieved with the function `RenesasErrorMessage()`.

 © 2007-2010 Analytica GmbH

Appendix B. I2C slave address formats

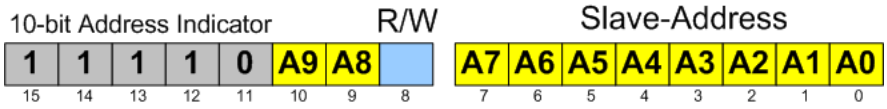
A standard I2C address is the first byte sent be the I2C master, whereas only the first seven bit form the adress, the last bit (R/W-bit) defines the direction in which the data is sent. I2C has a 7-bit address space and can address 112 slaves on a single bus (16 of the 128 addresses are reserved fo special purposes).

Figure B.1. Definition of a I2C slave address in 7-bit format

Each I2C-able IC has a determined bus address. The 4 upper bits of the bus address are called *Device Type Identifier* and define the chip type. The lowest three bits called sub-address or *Chip Enable Address* are usually defined by the corresponding wired control pins. So, in total up to 8 similar IC's can be used on a single I2C bus.

Because of a lack of address space a 10-bit addressing mode was introduced later. This new mode is downwards compatible to the 7-bit standard mode through usage of 4 of the 16 reserved addresses. Both addressing modes can be used simultaneously, which implies that 1136 slaves can be used on a single bus.

Figure B.2. Definition of a I2C slave address in 10-bit format



Note

Devices of type AnaGate SPI and AnaGate Universal Programmer do support both addressing modes in general. The I2CRead(3) and n>I2CWrite(3) functions address the slaves via a two byte parameter.

Addressing of serial EEPROM

The device type identifier of a serial EEPROM is defined as `0xA`. This results to the following schematic structure of an address (the chip enable bits are often named E0, E1 and E2 in literature):

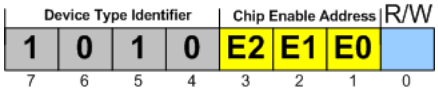


Table B.1. I2C EEPROM addressing examples

	Device Type Identifier				Chip Enable [1] [2]			R/W	EEPROM-Memory
	b7	b6	b5	b4	b3	b2	b1	b0	
M24C01	1	0	1	0	E2	E1	E0	R/W	128 byte
M24C02	1	0	1	0	E2	E1	E0	R/W	256 byte
M24C04	1	0	1	0	E2	E1	A8	R/W	512 byte

© 2007-2010 Analytica GmbH

	Device Type Identifier				Chip Enable [1] [2]			R/W	EEPROM-Memory
	b7	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**	
M24C08	1	0	1	0	E2	A9	A8	R/W	1024 byte
M24C16	1	0	1	0	A10	A9	A8	R/W	2048 byte
M24C64	1	0	1	0	E2	E1	E0	R/W	8192 byte

[1] E0,E1 and E2 are compared against the respective external pins on the memory device.

[2] A10, A9 and A8 represent most significant bits of the address.

© 2007-2010 Analytica GmbH

Appendix C. FAQ - Frequent asked questions

Here is a list of frequently asked questions.

C.1. Common questions

Q: No LAN connection

A: If no LAN connection is registered (the link LED next to the RJ45 socket does not light up), please check the wiring between the AnaGate and the hub or switch. You need to use a crossover cable when connecting the device directly to a PC.

C.2. Questions concerning AnaGate CAN

Q: What is the value of the termination resistor when the termination option of the device is activated?

A: The termination resistor of the AnaGate is driven by an FET transistor. The resistor itself has 110 Ohm while the internal resistance of the FET is 10 Ohm if the FET is activated. So the resulting resitance is 120 Ohm, as required by the CAN bus.

Q: Does Analytica offer a CAN gateway which do not have an galvanically isolated CAN interface.?

A: Any device that is actively connected to an CAN bus should be optical isolated. Especially when using USB-operated devices (like the AnaGate USB), it is essential to have an galvanically isolated device, because the device is power supplied by the PC.

C.3. Questions concerning AnaGate I2C

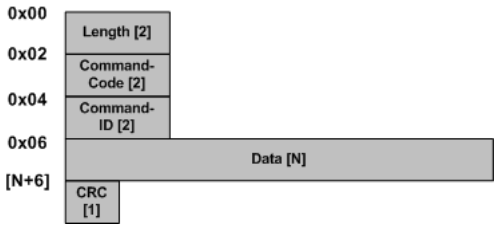
Q: What is the correct order to connect the GND / SCL and SDA when using an external power supply?

A: To avoid potential damage to the AnaGate I2C, the GND pin MUST be connected to the application board first. Only then can the SCL/SDA pins be allowed to make contact with the application board.

C.4. Questions concerning the communication protocol

Q: The calculation of the check sum (CRC) do not work!

A: The following figure illustrates the princible layout of an AnaGate telegram.



The checksum is defined as a byte calculated by XOR from all the existing bytes in an AnaGate telegram, excluding the length bytes and the CRC byte.

The following C code function computes a valid CRC of an already created command telegram.

© 2007-2010 Analytica GmbH

```
unsigned char CalcCRC( char * pBuffer, int nBufferLength )
{
   int i;
   unsigned char nCRC = pBuffer[2]; // skip the length bytes

   // XOR over all bytes in the message except the length information and the last byte
   for( i = 3; i < nBufferLength -1; i++ )
   {
      nCRC ^= pBuffer[i];
   }
   return nCRC;
}
```

When using the function `CalcCRC` the parameter *`pBuffer`* must point to the data buffer, which contains the already created complete data telegram. The length parameter *`nBufferLength`* depends on the created command type and can be computed as shown below:

```
buffer length = sizeof( command length ) + sizeof( command code )
                 + sizeof( command id ) + sizeof( CRC ) + sizeof(data)
              = 7 + sizeof(data)
```


© 2007-2010 Analytica GmbH

Appendix D. Technical support

The AnaGate hardware series and all existing programming interfaces are developed and supported by Analytica GmbH. Technical support can be requested as follows:

Internet

The AnaGate web site [http://www.anagate.de/en/index.html] of Analytica GmbH contains information and software downloads for AnaGate Library users:

- Product updates featuring bug fixes or new features are available here free of charge.

EMail

If you require technical assistance over the Internet, please send an e-mail to

`<support@analytica-karlsruhe.de>`

To help us provide you with the best possible support, please keep the following information and details at hand when you contact our Support Team.

- Version number of the AnaGate library
- AnaGate hardware series model and firmware version
- Name and version of the operating system you are using

© 2007-2010 Analytica GmbH

Bibliography

Books

Other publications

[NXP-I2C] NXP Semiconductors. Copyright © 2007 NXP Semiconductors. *UM10204*. I2C-bus specification and user manual. Rev. 03. 19.06.2007.

[TCP2010] Analytica GmbH. Copyright © 2010 Analytica GmbH. *Manual TCP-IP communication*. Version 1.2.6. 15.05.2008.